УДК 681. 3

М.К.Буза
Белорусский государственный университет, г. Минск, Беларусь
Беларусь, г. Минск, пр. Независимости, 4, bouza@bsu.by

АНАЛИЗ ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛЬНЫХ ТЕХНОЛОГИЙ

M.K.Bouza
Belarusian State University, Minsk, Belarus
Belarus, Minsk, Nezavisimosti ave., 4, bouza@bsu.by

ANALYSIS OF THE EFFECTIVENESS OF PARALLEL TECHNOLOGIES

М.К.Буза
Білоруський державний університет, м. Мінськ, Білорусь
Білорусь, м. Мінськ, пр. Незалежності, 4, bouza@bsu.by

АНАЛІЗ ЕФЕКТИВНОСТІ ПАРАЛЕЛЬНИХ ТЕХНОЛОГІЙ

В статье исследованы технологии MPI и OpenMP на предмет повышения скорости обработки данных. Представлен и проанализирован интегральный подход к построению параллельных технологий. Приведены оценки ускорения и эффективности используемых технологий раздельно и предложенной интегральной технологии.
Ключевые слова: общая память, параллельная технология, ускорение, эффективность, message passing interface, data race.

In the article the MPI and OpenMP technology for increasing the speed of data processing was investigated. Presented and analyzed an integrated approach to the construction of parallel technologies. The estimations of the acceleration and efficiency used separately and the proposed integrated technologies.
Key words: shared memory, parallel technology, acceleration, efficiency, message passing interface, data race.

У статті досліджено технології MPI і OpenMP на предмет підвищення швидкості обробки даних. Представлений і проаналізований інтегральний підхід до побудови паралельних технологій. Наведено оцінки прискорення та ефективності використовуваних технологій роздільно і запропонованої інтегральної технології.
Ключові слова: спільна пам'ять, паралельна технологія, прискорення, ефективність, message passing interface, data race.

Введение

Сегодня, благодаря развитию компьютерных технологий, стало возможным приступить к решению задач, требующих обработки больших объемов информации за разумное время.

Одним из наиболее популярных средств программирования для компьютеров с общей памятью, базирующихся на традиционных языках программирования, в настоящее время является технология OpenMP. Для создания параллельной версии последовательной программы пользователю предоставляется набор директив, функций и переменных окружения.

Для компьютеров с распределенной памятью используется технология MPI (*Message Passing Interface)*. Основным способом взаимодействия параллельных процессов в таких системах является передача сообщений. Стандарт MPI фиксирует интерфейс, который должен соблюдаться как системой программирования на каждой вычислительной платформе, так и пользователем при создании своих программ.

MPI-программа – это множество взаимодействующих процессов. Все процессы порождаются один раз, образуя параллельную часть программы. В ходе выполнения MPI-программы порождение дополнительных процессов или уничтожение существующих не допускается (в MPI-2.0 такая возможность появилась). Каждый процесс работает в своем адресном пространстве, никаких общих переменных или данных в MPI нет.

В статье рассматривается задача эффективности применения технологий MPI и OpenMP при решении разнообразных задач. Несмотря на то, что эти технологии работают с различными типами организации памяти (распределенной и общей) исследована возможность совместного использования технологий MPI+OpenMP для разработки параллельных программ и оценена ее эффективность на конкретном приложении.

Исследования показывают, что большинство узких мест при обработке параллельных программ может быть объединено в три группы. Первая связана с тем, что на различных параллельных компьютерах не все потоки данных обрабатываются идентично. Изучая особенности архитектуры компьютера, очень важно понять, что представляют собой те и другие потоки и описать их формально. Вторая определяется структурой связей между процессами программы. И, наконец, третья зависит от используемой в компиляторе технологии отображения программ в машинный код [1].

Система программирования MPI

Стандарт MPI фиксирует интерфейс, который должны соблюдать как система программирования MPI на каждой вычислительной системе, так и пользователь при создании своих программ. Интерфейс поддерживает создание параллельных программ в архитектуре MIMD (Multiple Instruction Multiple Data), что подразумевает объединение процессов с различными исходными текстами. Однако на практике программисты гораздо чаще используют SPMD-модель *(Single Program Multiple Data)*, в рамках которой для всех параллельных процессов используется один и тот же код. В настоящее время значительное количество реализаций MPI поддерживают работу с нитями.

Все дополнительные объекты: имена функций, константы, предопределенные типы данных и т. п., используемые в MPI, имеют префикс **mpi_**. Например, функция посылки сообщения от одного процесса другому имеет имя **MPi_send**.Все описания интерфейса MPI собраны в файле mpi.h, поэтому в начале MPI-программы должна стоять директива **#include <mpi.h>**.

Практически все программы, подготовленные с использованием коммуникационной технологии MPI, должны содержать средства не только для порождения и завершения параллельных процессов, но и для взаимодействия запущенных процессов между собой.

В *операциях коллективного взаимодействия* процессов участвуют все процессы коммуникатора. Соответствующая процедура должна быть вызвана каждым процессом, возможно со своим набором параметров. Возврат из процедуры коллективного взаимодействия может произойти в тот момент, когда участие процесса в данной операции уже закончено. Как и для блокирующих процедур, возврат означает то, что разрешен свободный доступ к буферу приема или посылки. Асинхронных коллективных операций в MPI нет.

Топология – это механизм сопоставления процессам некоторого коммуникатора альтернативной схемы адресации. В MPI топологии виртуальны, то

есть они не связаны с физической топологией коммуникационной сети. Топология используется программистом для более удобного обозначения процессов и, таким образом, приближения параллельной программы к структуре математического алгоритма. Кроме того, топология может использоваться системой для оптимизации распределения процессов по физическим процессорам параллельного компьютера при помощи изменения порядка нумерации процессов внутри коммуникатора.

Модель параллельной программы в OpenMP

Распараллеливание в OpenMP выполняется явно при помощи вставки в текст программы специальных директив, а также вызова вспомогательных функций. При ис-пользовании OpenMP предполагается *SPMD-модель* параллельного программирова-ния, в рамках которой для всех параллельных нитей используется один и тот же код [2].

Программа начинается с *последовательной области* – где работает один процесс (нить). При входе в *параллельную область* порождается ещё некоторое число процессов, между которыми в дальнейшем распределяются части кода. По завершении параллельной области все нити, кроме одной (нити-мастера), завершаются, и начинается последовательная область. В программе может быть любое количество параллельных и последовательных областей. Кроме того, параллельные области могут быть также вложенными друг в друга. В отличие от полноценных процессов, порождение нитей является относительно быстрой операцией, поэтому частые порождения и завершения нитей не сильно влияют на время выполнения программы.

Для разработки эффективной параллельной программы необходимо, чтобы все нити в программе были *равномерно загружены* полезной работой. Это достигается балансировкой загрузки, используя различные механизмы OpenMP.

Существенной является необходимость синхронизации *доступа к общим данным* [3]. Само наличие данных, общих для нескольких нитей, приводит к конфликтам при одновременном несогласованном доступе. Поэтому значительная часть функциональности OpenMP предназначена для осуществления различного рода синхронизаций работающих нитей.

Модель данных

Модель данных в OpenMP предполагает наличие как общей для всех нитей области памяти, так и локальной области памяти для каждой нити. Переменные в параллельных областях программы разделяются на два основных класса:

• shared *(общие;* все нити видят одну и ту же переменную);

• private *(локальные,* приватные; каждая нить видит свой экземпляр данной переменной).

Общая переменная всегда существует лишь в одном экземпляре для всей об-ласти действия и доступна всем нитям под одним и тем же именем. Объявление локальной переменной вызывает порождение своего экземпляра данной переменной (того же типа и размера) для каждой нити. Изменение нитью значения своей локальной переменной никак не влияет на изменение значения этой локальной переменной в других нитях.

Если несколько переменных одновременно записывают значение общей пе-ременной без выполнения синхронизации или если, как минимум, одна нить читает

значение общей переменной и, как минимум, одна нить записывает значение этой переменной без выполнения синхронизации, то возникает ситуация так называемой *«гонки данных» (data race),* при которой результат выполнения программы непредсказуем.

По умолчанию, все переменные, порождённые вне параллельной области, при входе в эту область остаются общими (shared). Исключение составляют переменные, являющиеся счетчиками итераций в цикле. Переменные, порождённые внутри параллельной области, по умолчанию являются локальными (private). Не рекомендуется постоянно полагаться на правила по умолчанию. Для большей надёжности лучше явно описывать классы используемых переменных, указывая в директивах OpenMP опции private, shared, firstprivate, lastprivate, reduction.

Динамически выделенная память также является общей, однако указатель на неё может быть как общим, так и локальным.

Критерии оценки эффективности

Для оценки параллельных алгоритмов в основном используют *ускорение* параллельных алгоритмов и их *эффективность*.

Ускорение показывает, во сколько раз применение параллельного алгоритма уменьшает время решения задачи по сравнению с последовательным алгоритмом. Оно определяется величиной

$$S_N = \frac{T_1}{T_N},$$

где T_1 – время выполнения алгоритма на одном процессоре, T_N –время выполнения того же алгоритма на N процессорах.

Очевидно, что идеальным является ускорение $S_N = N$. В реальности это ускорение недостижимо из-за отсутствия максимального параллелизма в алгоритме; несбалансированности загрузки процессоров; временных затрат на обмен данными, разрешение конфликтов и на синхронизацию.

В качестве оценки эффективности будем рассматривать величину

$$E_N = \frac{S_N}{N},$$

где S_N – ускорение параллельного алгоритма, N – количество процессоров в системе.

Несбалансированность загрузки процессоров существенно зависит от алгоритмов планирования [4].

Неизвестен эффективный способ априорного определения числа процессоров, для которого может быть получено наилучшее среди всех возможных распределение. Как правило, рассматриваются алгоритмы распределения при фиксированном числе процессоров. Задача может быть сформулирована следующим образом.

Пусть задано P-процессоров и множество задач $Z = \{Z_1, Z_2, ..., Z_n\}$, которые надо решить на ВС. Каждая задача из Z может характеризоваться некоторыми параметрами, в частности, объемом требуемой памяти, необходимым временем центрального процессора и т. д. Следует построить распределение $S_1, S_2, ..., S_p$ задач по процессорам ВС, учитывая параметры задач, с тем, чтобы минимизировать, скажем, общее время решения всего множества задач Z.

В общем случае распределение заданий по вычислителям необходимо осуществить таким образом, чтобы была решена одна из следующих задач:

- время решения всего множества задач было минимальным;
- загрузка всех компонент вычислительной системы была максимальной;
- выбор конфигурации системы был оптимальным при заданном времени решения всего множества задач.

При распределении множества задач следует учитывать существующие связи между ними и обеспечивать экстремальное значение некоторому критерию качества при определенных ограничениях. Достижение экстремума может производиться и за счет изменения структуры ВС. Однако, динамическая адаптация структуры ВС очень ограничена, и поэтому обычно считают, что состав и структура ВС не изменяются за все время ее работы.

Описание экспериментов

Для оценки ускорения и эффективности выберем умножение матриц и решение систем линейных алгебраических уравнений методом Гаусса.

Данные алгоритмы были реализованы на языке программирования C++ с использованием программного продукта Microsoft Visual Studio 2010 и библиотеки для создания параллельных программ MPICH2. Он тестировался на 4-ядерном процессоре Intel® Core™ i7-3770 (частота процессора – 3.4 ГГц, объем кэш-памяти – 8 Мб, объем оперативной памяти – 8 Гб). Количество потоков совпадает с количеством ядер, задействованных в вычислениях.

Таблица 1. Результаты тестирования алгоритма умножения матриц

Размер матриц	1 ядро	Параллельный алгоритм	
		2 ядра	4ядра
500	0,132132	0,068901	0,038424
1000	7,415507	3,829554	1,965694
2000	70,38636	35,93459	18,6018
3000	255,0253	130,3076	68,03396

Затраты на коммуникации определяются пропускной способностью коммуникационной сети, латентностью, ее диаметром, который в значительной мере определяется топологий коммуникационной сети. Коммуникационные затраты могут быть сокращены за счет использования встречных обменов данными и асинхронной передачи данных. Несложно видеть, что ускорение для двух процессов находится на уровне $E_N \approx 1{,}94$, а для четырех – на уровне $E_N \approx 3{,}75$.

Заметим, что при большой размерности задачи наблюдается небольшое снижение ускорения (это происходит и на двух, и на четырех процессах). Это связано со спецификой работы процессора. Для двух процессов эффективность

$S_N = 0{,}97$, для четырех – $S_N = 0{,}95$. Незначительное снижение эффективности связано с увеличением затрат на коммуникацию при увеличении количества процессов.

Таблица 2.Результаты тестирования параллельного алгоритма Гаусса

N	1 поток	2 потока	4 потока
1000	0,94	0,48	0,28
2000	7,82	4,2	3,01
3000	26,72	14,27	10,98
4000	63,77	33,9	26,18
5000	130,09	68,19	51,45

Время работы алгоритмов указано в секундах.

Для исследования интегральной технологии (MPI+OpenMP) рассмотрим задачу моделирования процесса роста дендритных кристаллических структур. При этом для организации взаимодействия между узлами вычислительной системы используются средства MPI, а для обеспечения эффективных вычислений в пределах отдельных многопроцессорных узлов с общей памятью применяется OpenMP.

Для проведения эксперимента рассмотрим алгоритм моделирования процесса роста дендритных кристаллических структур [5].

Область, в которой рассматривается образование кристалла, разбивается сеткой на элементарные квадратные ячейки. Одну ячейку этой сетки мы будем считать минимальной неделимой единицей пространства – элементарным объемом. Каждая элементарная ячейка характеризуется своим состоянием: раствором или кристаллом, находящимся в этой ячейке, концентрацией вещества в ячейке, концентрацией примеси в ячейке.

Рост грани кристалла рассматривается как пуассоновский поток событий. Событием является кристаллизация элементарной ячейки. Вероятность кристаллизации *p* элементарной ячейки в зависимости от шага по времени A_t вычисляется по формуле (1).

$$p = 1 - \exp\left(-V\frac{\Delta t}{\Delta x}\right) \qquad (1)$$

где *V* – средняя скорость роста грани кристалла; Δt, Δx – соответственно шаги дискретизации по времени и пространству. За один шаг по времени кристаллизуется одна ячейка, выбранная случайно среди ячеек с одинаковой вероятностью кристаллизации, равной максимальной.

Скорость роста грани кристалла зависит, в свою очередь, от концентрации примеси. Концентрация примеси в растворе $C_в$ может меняться от нуля до плотности этого вещества в твердой фазе $p_в$. Очевидно, что скорость роста грани равна нулю при нулевой концентрации и стремится к бесконечности при приближении концентрации к максимальной $p_в$. Концентрация насыщенного раствора соли C_0 и базовая скорость роста кристалла V_0 являются известными величинами [6]. Окончательные выражения для скорости роста имеют вид (2).

$$V = V_0 \frac{C_e}{C_0}\left(\frac{\rho_e - C_0}{\rho_e - C_e}\right) \quad (2)$$

При совмещении двух технологий особых проблем не возникло, так как каждая из них затрагивает разные аспекты алгоритма. MPI используется для разделения и передачи данных, а OpenMP для распределения операций с полученными данными.

Для сравнения эффективности работы программы было проведено ряд экспериментов при различных конфигурациях вычислительных систем (рисунок 1): на одном ядре, на двух ядрах одного процессора, на вычислительных станциях с двухъядерным процессором Celeron Dual-Core 2*2.0 GHz, 4 Gb RAM, операционной системой Windows 7 x64, соединённых по сети. А также на рабочей станции с четырёхъядерным процессором Intel Xenon 2.0 GHz, 4 Gb RAM, операционной системой Windows 7 x64.

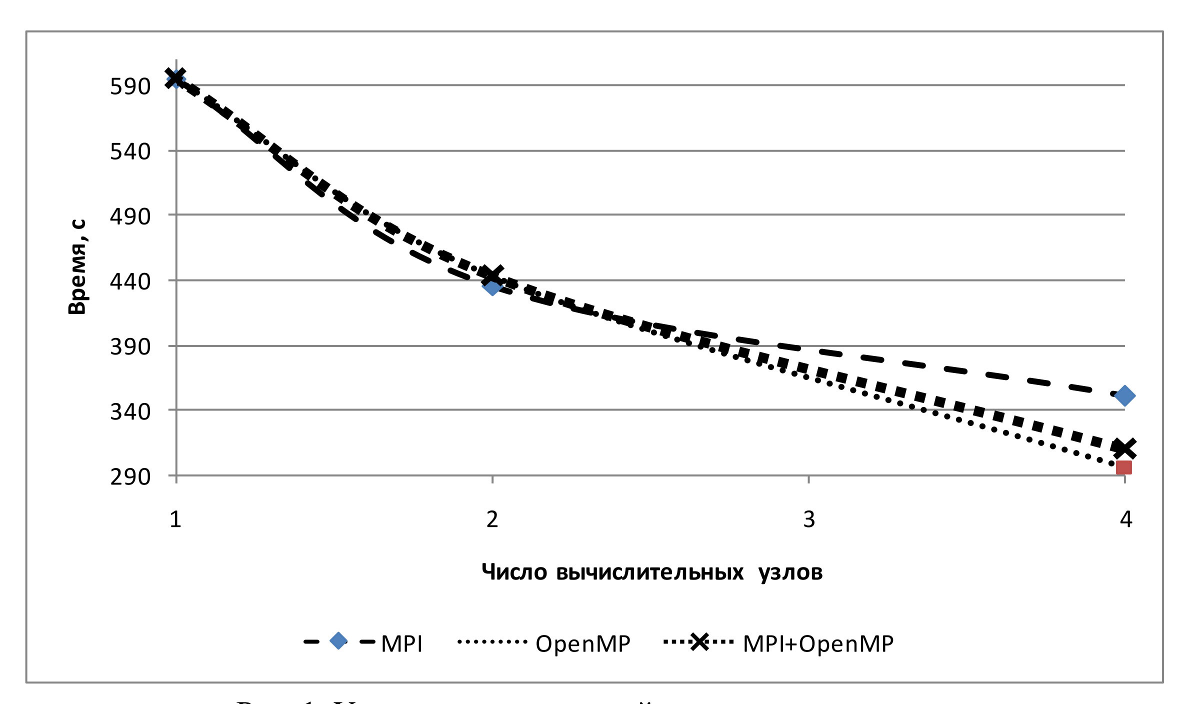


Рис. 1. Ускорение вычислений на разном количестве узлов

В таблице 3 представлены результаты работы алгоритма при размере сетки 1500x1500. Из этих данных видно, что при разделённой памяти и сильной зависимости смежных потоков по данным происходит замедление вычислений по сравнению с системами с общей памятью. Это связано с тем, что тратится время на пересылку данных, во время которого вычислительная часть программы простаивает.

Таблица 3. Результаты запуска распараллеленной программы на разном числе вычислительных узлов

Число узлов	Время, с
1	595,2
2 процессора (MPI)	435,6
2 ядра (OpenMP)	443,7
4 процессора (MPI)	350,8
4 ядра (OpenMP)	296,4
2 процессора по 2 ядра (MPI + Open MP)	310,2

В результате анализа полученных данных можно заключить, что при организации параллельных вычислений на многопроцессорных системах с разным типом памяти, можно применять комбинированные технологии разработки параллельных программ.

Заключение

Анализ технологий и результатов экспериментов показал, что несмотря на существенное различие в типах требуемой памяти технологии MPI и OpenMP не только можно, но и следует использовать совместно. Важно лишь правильно распределить при решении конкретной задачи, на каком этапе ее решения использовать ту или иную технологию. От этого существенно зависит ускорение при ее решении.

Литература

1. Воеводин, В. Параллельные вычисления / В. Воеводин, Вл. Воеводин. – СПб.: БХВ-Петербург, 2002. – 608 с.
2. Антонов, А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие /А.С. Антонов. - М.: Изд-во МГУ, 2004. - 71 с.
3. Буза, М.К. Проектирование программ для многоядерных процессоров. / М.К.Буза, О.М.Кондратьева. - Минск: БГУ, 2013. – 48 с.
4. Буза, М.К. Системы параллельного действия / М.К.Буза. – Минск: БГУ, 2009. - 415 с.
5. Вайнштейн, Б. Современная кристаллография: Структура кристаллов. / Б. Вайнштейн, В.Фридкин, В. Инденбом. - М., Наука. 1979. 360 с.
6. Саратовкин, Д. Дендритная кристаллизация / Д. Саратовкин. М., Металлургиздат. 1953. - 95 с.

Literatura

1. Voyevodin, V. Parallelnye vychisleniya / V. Voyevodin, Vl. Voyevodin. – SPb.: BKhV-Peterburg, 2002. – 608 s.
2. Antonov, A.S. Parallelnoye programmirovaniye s ispolzovaniyem tekhnologii MPI: Uchebnoye posobiye /A.S. Antonov. - M.: Izd-vo MGU, 2004. - 71 s.
3. Buza, M.K. Proyektirovaniye programm dlya mnogoyadernykh protsessorov. / M.K.Buza, O.M.Kondratyeva. - Minsk: BGU, 2013. – 48 s.
4. Buza, M.K. Sistemy parallelnogo deystviya / M.K.Buza. – Minsk: BGU, 2009. - 415 s.
5. Vaynshteyn, B. Sovremennaya kristallografiya: Struktura kristallov. / B. Vaynshteyn, V. Fridkin, V. Indenbom. - M., Nauka. 1979. 360 s.
6. Saratovkin, D. Dendritnaya kristallizatsiya / D. Saratovkin. M., Metallurgizdat. 1953. - 95 s.

RESUME

M.K.Bouza

Analysis of the effectiveness of parallel technologies

In the article the MPI and OpenMP technology for increasing the speed of data processing was investigated. Presented and analyzed an integrated approach to the construction of parallel technologies. The problems arising when processing parallel programs are formulated. Coast of communications is offered to be reduced due to use of counter data exchanges, and also their asynchronous transfer. The estimations of the acceleration and efficiency used separately and the proposed integrated technologies.

Key words: shared memory, parallel technology, acceleration, efficiency, message passing interface, data race.

Поступила в редакцию 31.08.2015

